МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

отчет

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Стек, очередь, дек

Студент гр.8304 _____ Ястребов И.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Задание.

Вариант 11-а-в

Рассматривается выражение следующего вида:

< выражение >::= < терм > | < терм > + < выражение > |

< терм > - < выражение >

< терм > ::= < множитель > | < множитель > * < терм >

< множитель > ::= < число > | < переменная > | (< выражение >) |

< множитель > ^ < число >

< число > ::= < цифра >

< переменная > ::= < буква >

Требуется:

а) вычислить как целое число значение выражения (без переменных), записанного в постфиксной форме в заданном текстовом файле postfix;

Цель работы.

Решить полученную задачу, реализовав стек на основе массива.

Описание алгоритма.

Программа создает обьект реализованного класса стека на массиве — далее , анализируя входную строку, на стеке хранятся и извлекаются текущие значения арифметических подвыражений. Используется стандартный алгоритм вычисления арифметического выражения, записанного в постфиксной форме. Подробное описание : в качестве stack pointer“a используется int index — номер текущего последнего заполеннного адреса в массиве. Память выделяется с помощью new. Алгоритм вычисления ариф метического значения : программа идет по строке, все числа последовательно кладутся на стек, как только встречается знак операции, программа применяет эту операцию к двум верхним элементам стека, после чего кладет вместо них результат операции. В конце на стеке остается одно число — конечное значение арифметического выражения.

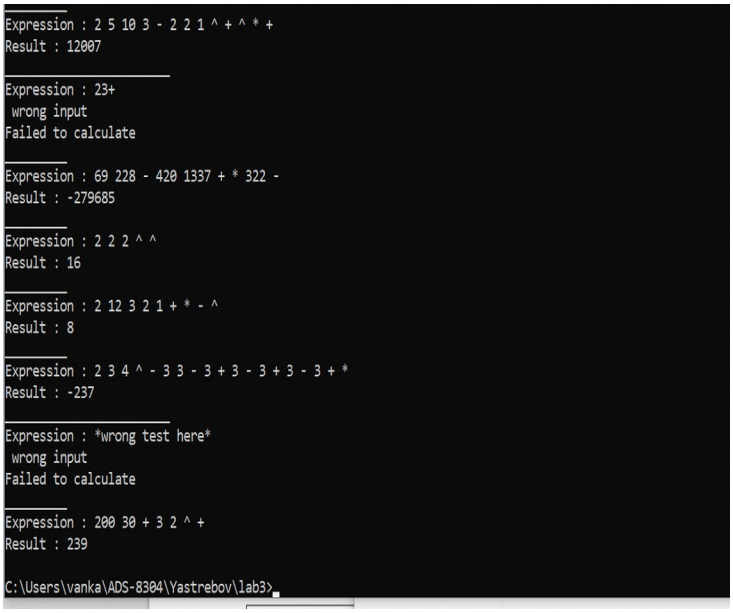
Описание функций и структур данных.

- `Class Vec_stack;` - класс, реализация стека на основе динамического массива, с методами push, pop, isempty, top
- `EvaluateExpression(std::string &expression)`- функция, вычисляющая значения постфиксного выражения
- `IsOperator(char &c)` определяет, является ли символ знаком операции
- `PerformOperation(char&, int&, int&)` совершает арифметическую операцию над двумя операндами

Выводы.

В ходе выполнения данной работы был написан класс, стек на основе динамического массива.

Тестирование



Пример вывода программы

Исходный код

```
/*
11 Рассматривается выражение следующего вида:
< выражение >::= < терм > | < терм > + < выражение > | < терм > - < выражение >
< терм > ::= < множитель > | < множитель > * < терм >
< множитель > ::= < число > | ( < выражение > ) | < множитель > ^ < число >
< число > ::= < цифра >
*/


#include "pch.h"
#include <iostream>
#include <string>
#include <fstream>
#include "Vec_Stack.hpp"


nInt EvaluatePostfix(std::string &expression);

nInt PerformOperation(char &operation, int &operand1, int &operand2);
```

```
bool IsOperator(char &C);

int main(int argc, char* argv[])
{
std::ifstream input;

if(argc == 1)
input.open("postfix.txt");

else
input.open(argv[1]);

if (!input)
{
std::cout << "Couldn't open source file" << std::endl;

return 0;
}

std::string expression;



while (std::getline(input, expression)) {

nInt result = EvaluatePostfix(expression);

if (result != std::nullopt)
      std::cout << "_________\nExpression : " << expression << "\nResult : " <<
result.value() << std::endl;

else
      std::cout << "Failed to calculate" << std::endl;
}

return 0;
}

nInt EvaluatePostfix(std::string &expression)
{
Vec_Stack vecStack;

for (size_t i = 0; i < expression.size(); ++i)
```

```
{
if (expression[i] == ' ')
	continue;

if (IsOperator(expression[i])) {
	nInt rightOperand = vecStack.pop();

	if (rightOperand == std::nullopt) {
		std::cout << "YOU DIED" << std::endl;

		return std::nullopt;
	}

	nInt leftOperand = vecStack.pop();

	if (leftOperand == std::nullopt) {
		std::cout << "YOU DIED" << std::endl;

		return std::nullopt;
	}

	nInt result = PerformOperation(expression[i], leftOperand.value(),
rightOperand.value());

	if (result == std::nullopt)
		return std::nullopt;

	else
		vecStack.push(result.value());
}

else if (isdigit(expression[i])) {
	int operand(0);

	while (i < expression.size() && isdigit(expression[i])) {
		operand = operand * 10 + (expression[i] - '0');

		++i;
	}

	--i;

	vecStack.push(operand);
```

```
}

else {
	std::cout << "Incorrect input" << std::endl;

	return std::nullopt;
}
}

return vecStack.pop();
}

bool IsOperator(char &c)
{
return (c == '+' ||
c == '-' ||
c == '*' ||
c == '^');
}

nInt PerformOperation(char &operation, int &operand1, int &operand2)
{
if (operation == '+')
return operand1 + operand2;

else if (operation == '-')
return operand1 - operand2;

else if (operation == '*')
return operand1 * operand2;

else if (operation == '^')
return (int)pow(operand1, operand2);

else
return std::nullopt;
}
```

Vec_Stack.hpp :

```
#pragma once
#include <optional>
constexpr auto DEFAULT_SIZE = 1<<8;
```

```
typedef std::optional<int> nInt;

class Vec_Stack {
private:
	int* stack;
	size_t index;
	size_t size;

public:
	Vec_Stack(size_t startSize = DEFAULT_SIZE) {
		stack = new int[startSize];
		size = startSize;
		index = -1;
	}

	void push(int element) {

		if (index == (size - 1))
		{
			int* newStack = new int[size + DEFAULT_SIZE];

			std::copy(&stack[0], &stack[size], &newStack[0]);

			delete[] stack;

			stack = newStack;

			size += DEFAULT_SIZE;

			stack[++index] = element;
		}

		else {
			stack[++index] = element;
		}
	}
```

```
	nInt pop() {
		if (index == -1)
			return std::nullopt;
		else
			return stack[index--];
	}

	bool isEmpty() {
		return (index == -1);
	}

	nInt top() {
		if (index != -1)
			return stack[index];
		else
			return std::nullopt;
	}

	~Vec_Stack() {
		delete[] stack;
	}
};
```